UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08008 INFORMATICS 2A: PROCESSING FORMAL AND NATURAL LANGUAGES

Monday 9 th December 2013

09:30 to 11:30

INSTRUCTIONS TO CANDIDATES

1. **Answer all five questions in Part A, and two out of three questions in Part B. Each question in Part A is worth 10% of the total exam mark; each question in Part B is worth 25%.**
2. **Use a single script book for all questions.**
3. **Calculators may be used in this exam.**

Convener: J. Bradfield
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

PART A
ANSWER ALL QUESTIONS IN PART A.

1. (a) Explain briefly what is meant by a *language processing pipeline*. List, in order, the stages in a typical such pipeline (i) for the Java programming language, and (ii) for spoken English. You need not mention all possible stages in the pipeline, but should include at least four important stages in each list. [*6 marks*]

   (b) Mention two important differences in nature between formal and natural languages. Briefly explain what impact these differences have on the way such languages must be processed. [*4 marks*]

2. For each $n \geq 1$ consider the language L_n, over the alphabet $\Sigma = \{a, b, c\}$, defined below.

$$L_n \;=\; \{x \in \Sigma^* \mid |x| \geq n \text{ and the } n\text{th symbol from the end of } x \text{ is } a\}$$

   For example, the strings *aa*, *ab* and *bac* are in L_2, but *a* and *aca* are not. And the string *cabcb* is in L_4, but *caba* is not.

   (a) Draw an NFA that recognises the language L_4. [*4 marks*]

   (b) Draw a minimal DFA recognising the language L_2. [*5 marks*]

   (c) For $n \geq 1$, how many states does a minimal DFA for L_n have? [*1 mark*]

3. Consider the following simple context-free grammar for commands in English. The start symbol is Com.

Com	→	give NP NP
NP	→	DT Nom
Nom	→	N | A N
DT	→	the | a
N	→	dog | bone
A	→	large | small

   (a) Convert this grammar to *Chomsky Normal Form* (CNF). You need not write out the productions with left hand side DT, N or A, which will remain unchanged. [*4 marks*]

   (b) With respect to your CNF grammar, draw a CYK parse chart for the command:

give the dog a bone

   Include pointers to show how each compound phrase is built from smaller phrases. [*6 marks*]

4. In the following context-free grammar, each rule is equipped with a semantic attachment showing how the corresponding phrases may be translated to lambda-expressions involving logical symbols and connectives. Note that *Rains* and *Snows* are propositional constants (i.e. predicate symbols with no arguments). The start symbol is BigS.

BigS →	S if S	{ $\mathsf{S}_2.\mathsf{Sem} \Rightarrow \mathsf{S}_1.\mathsf{Sem}$}
S →	it WeatherV	{ WeatherV.Sem}
WeatherV →	rains	{ *Rains* }
WeatherV →	snows	{ *Snows* }
S →	there is NP	{ $\exists$x. NP.Sem(x) }
NP →	a Nom	{ Nom.Sem}
Nom →	A N	{ λy. A.Sem(y) $\wedge$ N.Sem(y) }
A →	serious	{ λz. *Serious*(z) }
N →	problem	{ λz. *Problem*(z) }

(a) Draw the parse tree for the following sentence, allowing plenty of room for annotations:

there is a serious problem if it snows

Starting from the leaves of the tree and working upwards, annotate each node with the raw lambda-expression assigned to it by the above semantics. [*6 marks*]

(b) Show how the lambda-expression associated with the complete sentence above β-reduces in several steps to a logical expression in normal form. You should show each β-reduction step explicitly. [*4 marks*]

5. (a) Consider the following noncontracting grammar, with terminals $\Sigma = \{\mathsf{a}, \mathsf{b}, !\}$; nonterminals S (the start symbol), T, A and B; and productions

S → ! T	a A → A a
T → A a | B b | A a T | B b T	a B → B a
! A → a !	b A → A b
! B → b !	b B → B b

Give a full derivation of:

a b ! a b

[*7 marks*]

(b) Give a concise mathematical description of the language generated by the grammar in part (a). [*2 marks*]

(c) At which level does this language reside in the Chomsky hierarchy? [*1 mark*]

PART B
ANSWER TWO QUESTIONS FROM PART B.

6. (a) Consider the following list of different types of machine.

   i. Deterministic finite automata
   ii. Nondeterministic pushdown automata
   iii. Nondeterministic linear bounded automata
   iv. Turing machines

   For each machine type in the above list, name the class of languages recognised by machines of that type, **and** give an example of a language in the class that cannot be recognised by any machine type higher up the list. [*8 marks*]

   (b) Consider a pushdown automaton (PDA) with two control states $Q = \{q1, q2\}$, start state $q1$, input alphabet $\Sigma = \{a, b\}$, stack alphabet $\Gamma = \{a, b, \perp\}$ (where $\perp$ is the start symbol), and transition relation:

$$
\begin{array}{lll}
q1 \xrightarrow{a,\perp:a\perp} q1 & q1 \xrightarrow{b,\perp:b\,b\perp} q1 & q2 \xrightarrow{\epsilon,\perp:b\perp} q1 \\
q1 \xrightarrow{a,a:a\,a} q1 & q1 \xrightarrow{b,a:\epsilon} q2 & q2 \xrightarrow{\epsilon,a:\epsilon} q1 \\
q1 \xrightarrow{a,b:\epsilon} q1 & q1 \xrightarrow{b,b:b\,b\,b} q1 & q1 \xrightarrow{\epsilon,\perp:\epsilon} q1
\end{array}
$$

   The automaton accepts on empty stack. (In the above description, we use the general notation

$$q \xrightarrow{s,x:\alpha} q'$$

   to mean that when the automaton is in control state $q \in Q$ and $x \in \Gamma$ is popped from the top of the stack, the input symbol or empty string $s \in \Sigma \cup \{\epsilon\}$ can be read to reach control state $q' \in Q$ with $\alpha \in \Gamma^*$ pushed onto the stack.)

   Describe in detail an execution of the above PDA that accepts the string

$$aaabba$$

   [*8 marks*]

   (c) Give a concise mathematical definition of the language L recognised by the PDA above. [*2 marks*]

   (d) Prove that the language L, defined in your answer to (c) above, cannot be recognised by any deterministic finite automaton. [*7 marks*]

7. The following is part of a grammar for generating simple cookery instructions:

$$\begin{array}{rcl} \mathsf{S} & \rightarrow & \text{place } \mathsf{NP}\ \ \mathsf{PP} \mid \text{remove } \mathsf{NP} \text{ from } \mathsf{NP} \\ \mathsf{NP} & \rightarrow & \mathsf{Noun} \mid \mathsf{NP}\ \ \mathsf{PP} \\ \mathsf{PP} & \rightarrow & \mathsf{Prep}\ \ \mathsf{NP} \end{array}$$

The start symbol is S. The terminal symbols Noun and Prep stand for word classes as follows:

$$\begin{array}{rl} \mathsf{Noun}: & \{ \text{ chips, burgers, tray, plate, oven } \} \\ \mathsf{Prep}: & \{ \text{ in, on, with } \} \end{array}$$

(a) Draw all possible parse trees for the phrase:

place chips on tray in oven

[*2 marks*]

(b) Treating the symbols Noun and Prep as terminals, use the *Earley algorithm* to parse the sequence

place Noun Prep Noun

You should show the execution of the algorithm as a table with a row for each step. Each row should include the start and end position of the portion of input processed, and a letter P, S or C to indicate whether the step is due to the predictor, scanner or completer. To get you started, the first row of your table should be

S → • place NP PP [0,0] P

You need not include all possible steps, but you should include all steps that contribute to a successful parse, and at least two that do not. You should explicitly mark the latter steps as 'spurious'. [*9 marks*]

(c) One possible method for resolving ambiguities in context-free grammars is to attach a probability weighting to each production rule, and then select the parse tree with the highest overall probability. Could this approach be used to select a preferred parsing for the above phrase? Briefly justify your answer. [*2 marks*]

(d) Another possible way to resolve ambiguities is to rewrite the grammar. Design an LL(1) grammar that is equivalent to the one above (again treating Noun and Prep as terminals). You may introduce new non-terminal symbols if they are helpful. [*5 marks*]

(e) Draw up the LL(1) parse table for your grammar. (You need not exhibit the First and Follow sets as part of your solution.) [*7 marks*]

8. In this question, we will consider an example of POS tagging using the following six tags, representing singular and plural nouns, third person singular and plural present-tense verbs, adjectives and adverbs:

Ns Np Vs Vp Adj Adv

We will also consider six *words*, to each of which is associated a *stem* and two or more *parts of speech* as follows:

Word	Stem	Parts of speech
fast	fast	Ns, Vp, Adj, Adv
fasts	fast	Np, Vs
hold	hold	Ns, Vp
holds	hold	Np, Vs
hook	hook	Ns, Vp
hooks	hook	Np, Vs

(a) Draw the state diagram for a non-deterministic *finite state transducer* which accepts as input any of the above six words (as a sequence of letters) followed by the word boundary marker #, and which can produce as output the corresponding stem followed by any of the possible POS tags for the word. For example, given the input string 'hold#', the transducer should be capable of producing either 'hold Ns' or 'hold Np' as output (and nothing else).

The input alphabet for the transducer should be { a,...,z,# }, while the output alphabet should be { a,...,z } together with the set of six POS tags (we regard each POS tag as a single output symbol).

Finally, your transducer should have as few states as possible for the task it performs. However, you will not be penalized heavily if your transducer has just a few more states than necessary. [*11 marks*]

(b) We will now work towards tagging the sequence:

hook holds fast

In this part of the question, we shall consider the simpler problem of tagging the corresponding sequence of *stems*:

hook hold fast

using just the reduced tagset N, V, Adj, Adv.

Use the Viterbi algorithm to tag this sequence, using the following transition and emission probabilities:

	to N	to V	to Adj	to Adv
from start	0.6	0.2	0.1	0.1
from N	0.4	0.3	0.1	0.2
from V	0.3	0.2	0.2	0.3
from Adj	0.5	0.1	0.3	0.1
from Adv	0.2	0.5	0.1	0.2

Transitions

	fast	hold	hook
N	0.2	0.3	0.5
V	0.2	0.5	0.3
Adj	1.0	0	0
Adv	1.0	0	0

Emissions

Show your working, and include explicit backtrace pointers in your Viterbi matrix. [*10 marks*]

(c) Explain how the techniques from parts (a) and (b) can be combined to produce taggings for word sequences using our original set of six tags. Illustrate the method by tagging the sequence:

hook holds fast

[*4 marks*]